Fixed Height Queries Tree Permutation Index for Proximity Searching

Karina Figueroa[1](✉), Rodrigo Paredes[2], J. Antonio Camarena-Ibarrola[1], and Nora Reyes[3]

[1] Universidad Michoacana, Morelia, Mexico
karina@fismat.umich.mx, camarena@umich.mx
[2] Universidad de Talca, Talca, Chile
raparede@utalca.cl
[3] Universidad Nacional de San Luis, San Luis, Argentina
nreyes@unsl.edu.ar

Abstract. Similarity searching consists in retrieving from a database the objects, also known as nearest neighbors, that are most similar to a given query, it is a crucial task to several applications of the pattern recognition problem. In this paper we propose a new technique to reduce the number of comparisons needed to locate the nearest neighbors of a query. This new index takes advantage of two known algorithms: FHQT (Fixed Height Queries Tree) and PBA (Permutation-Based Algorithm), one for low dimension and the second for high dimension. Our results show that this combination brings out the best of both algorithms, this winner combination of FHQT and PBA locates nearest neighbors up to four times faster in high dimensions leaving the known well performance of FHQT in low dimensions unaffected.

1 Introduction

Similarity searching consists in retrieving the most similar objects from a database to a given query. This problem is also known as nearest neighbor searching, which is a crucial task to several areas such as multimedia retrieval (i.e. images), computational biology, pattern recognition, etc. The similarity between objects can be measured with a distance function, usually considered expensive to compute, defined by experts in a specific data domain. Thus, the main objective of several proposed indexes is to reduce the number of distance evaluations to get the most similar objects in a database with respect to a given query.

Similarity searching can be mapped into a metric space problem. It can be seen as a pair $(\mathbb{X}, d)$, where $\mathbb{X}$ is the universe of objects and d is the distance function $d : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^+ \cup \{0\}$. The *distance* satisfies, for all $x, y, z \in \mathbb{X}$, the following properties: reflexivity $d(x, y) = 0$ iff $x = y$, symmetry $d(x, y) = d(y, x)$, and triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$. In practical applications, we have a working database $|\mathbb{U}| = n$, $\mathbb{U} \subseteq \mathbb{X}$.

Basically, there are two kinds of queries: range query (q, r) and k-nearest neighbor query $kNN(q)$. The first one retrieves all the objects within a given

J.A. Carrasco-Ochoa et al. (Eds.): MCPR 2017, LNCS 10267, pp. 74–83, 2017.
DOI: 10.1007/978-3-319-59226-8_8

radius r measured from q, that is, $(q,r) = \{u \in \mathbb{U}, d(q,u) \leq r\}$. The other one retrieves the k objects in $\mathbb{U}$ that are the closest to q. Formally, $|kNN(q)| = k$, and $\forall\ u \in kNN(q), v \in \mathbb{U},\ d(u,q) \leq d(v,q)$.

Several algorithms [8,10,11] have been proposed in order to answer these kind of queries. Examples of indexes that are relevant to this work are Fixed Queries Tree (FQT) [2] and the Permutation-Based Algorithm (PBA) [5]. Some indexes suffer the well known *curse of dimensionality* (that is, the searching effort increases as the dataset intrinsic dimensionality grows) [8]. In practice, there are indexes more adequate to certain dimensionality. For instance, the FQT is well suited for low dimensionality and the PBA for medium to high dimensionality. As the distance is considered expensive to compute, in this work we measure any cost in terms of the number of distance computations needed.

In this paper, we introduce an improvement on top of these two classic metric spaces searching algorithms: a variant of FQT (Fixed Height Queries Tree - FHQT) and PBA. This new index works well in both low and high dimensionality, because adequately combines the best of them.

The organization of this paper is as follows. Section 2 presents the basic concepts about metric spaces algorithms. Section 3 describes in detail the indexes FHQT and PBA, that are the base of our work. Section 4 introduces our proposal in details (indexing and searching). In Sect. 5, we experimentally prove our claims using both synthetic and real world datasets, the first one help us to known the performance of the parameters. In the last section we arrive to some conclusions and a discussion of future work.

2 Basic Concepts

Firstly, an algorithm aims to establish some structure or index over the database $\mathbb{U}$; then, when a query is given, the algorithm uses this structure to speed up the response time. Of course, in order to traverse through the index some distances computations are needed. This process obtains a set of non-discarded objects, then the query is compared with all these objects to answer the similarity query.

The answer to a similarity query can be *exact* or *approximate.* An approximate similarity searching is appealing when we require efficiency and instead we accept to lose some accuracy. This is specially relevant when we work on high intrinsic dimensionality spaces. Algorithms that obtain exact answers can be classified in two groups, namely pivots-based algorithms (PB) and compact-partitions algorithms (CP). While PB algorithms work well in low dimensions, CP algorithms work better in high dimensions. On the other hand, a good approach to solve similarity queries in an approximated fashion is to use the PBA, that are unbeatable in high and very high dimensions.

Pivot-Based Algorithms. A pivot-based algorithm chooses a set of *pivots* $\mathbb{P} = \{p_1, p_2, \ldots, p_j\} \subseteq \mathbb{U}, j = |\mathbb{P}|$. For each database element $u \in \mathbb{U}$, the PB algorithm computes and stores all the distances between u and the members of $\mathbb{P}$. The resulting set of distances $\{d(p_1, u), d(p_2, u), \ldots, d(p_j, u)\}, \forall u \in \mathbb{U}$ is used for building the index.

For a range query (q, r), the distances $d(p_i, q)$ $\forall p_i \in \mathbb{P}$ are computed. By virtue of the triangle inequality property, for each $u \in \mathbb{U}$, it holds that $\max_{1 \leq i \leq j} |d(q, p_i) - d(p_i, u)| \leq d(q, u)$. Therefore, if an element u satisfies that $r < \max_{1 \leq i \leq j} |d(q, p_i) - d(p_i, u)|$, then u can be safely discarded. Finally, every non-discarded element is directly compared with q and reported if it fulfills the range criterion.

Compact Partition Algorithms. A compact partition algorithm exploits the idea of dividing the space in compact zones, usually in a recursive manner, and storing a representative object (a "center") c_i for each zone plus a few extra data that permits quickly discarding the zone at query time. During search, entire zones can be discarded depending on the distance from their cluster center c_i to the query q. Two criteria can be used to delimit a zone, namely hyperplane and covering radius.

3 Related Works

In this section we describe the two algorithms that we use as base of our work: Fixed Height Queries Tree and Permutation-based Algorithm.

3.1 Fixed Height Queries Tree

The FHQT belongs to a family of indexes, all of them with about the same efficiency in terms of number of distances computed. These indexes are known as Fixed-Queries Tree (FQT) [2], Fixed Height FQT (FHQT) [1,2], Fixed-Queries Array (FQA) [7], and Fixed Queries Trie (FQTrie) [4]. All of them partition $\mathbb{U}$ in classes according to a distance, or range of distances, to a pivot p. In particular, the FHQT is built as follows. Firstly, a set of j pivots is chosen, the ranges of distances are stablished, and we associate a label l to each range. We start with pivot p_1 and for every range of distance l we pick the objects whose distance to p_1 belongs to the range labeled as l. For every non-empty subset, a tree branch with label l is generated. Next, we repeat the process recursively using the next pivot. Each pivot is used for all subtrees in the same level; therefore, pivot p_2 is used for all subtrees in the second level, p_3 for the third level and so on. The height of the tree is j.

For a given query q and radius r, $d(q, p_1)$ is computed and all branches whose range of distance do not intersect with $[d(q, p_1) - r, d(q, p_1) + r]$ are discarded. The process is repeated recursively for those branches not yet discarded using the next pivot. Hence, a set of candidates elements is obtained. Finally, all the candidates are directly compared with q to answer the similarity query.

3.2 Permutation-Based Algorithm (PBA)

A permutation-based algorithm can be described as follows: Let $\mathbb{P} \subset \mathbb{U}$ be a set of permutants with m members. Each element $u \in \mathbb{U}$ induces a preorder $\leq_u$ given by the distance from u towards each permutant, defined as $y \leq_u z \Leftrightarrow d(u, y) \leq d(u, z)$, for any pair $y, z \in \mathbb{P}$.

Let $\Pi_u = i_1, i_2, \ldots, i_m$ be the permutation of u, where permutant $p_{i_j} \leq_u p_{i_{j+1}}$. Permutants at the same distance take an arbitrary but consistent order. For every object in $\mathbb{U}$, its preorder of $\mathbb{P}$ is computed and associated to a permutation. The resulting set of permutations conforms the index, since a PBA does not store any distance.

Given the query $q \in \mathbb{X}$, the PBA search algorithm computes its permutation Π_q and compares it with all the permutations stored in the index. Then, the dataset $\mathbb{U}$ is traversed in increasing dissimilarity of permutations, comparing directly the objects in $\mathbb{U}$ with the query using the distance d of the particular metric space. There are many similarity measures between permutations. One of them is the L_s family of distances, that obeys Eq. (1), where $\Pi^{-1}(i_j)$ denotes the position of permutant p_{i_j} within permutation Π.

$$L_s(\Pi_u, \Pi_q) = \sum_{1 \leq j \leq |\mathbb{P}|} |\Pi_u^{-1}(i_j) - \Pi_q^{-1}(i_j)|^s \tag{1}$$

There are some special values for s. If $s = 2$ the distance is known as *Spearman Rho* (S_ρ); and for $s = 1$ it, is called *Spearman Footrule* (S_f).

4 Fixed Height Queries Tree Permutation (FHQTP)

We propose a new efficient metric index as result of a clever combination of the best features of the two aforementioned indexes. This way, at search time we can take advantage of both the search pruning of the FHQT and the prediction capability of the PBA. We use the same pivots of the FHQT as the permutants of our PBA, so that we produce the PBA index with no extra distance computation.

4.1 Index Construction

The first stage of the FHQTP consists in building the classic FHQT, maintaining all the distances computed during the process. As we have computed the real distances between every object $u \in \mathbb{U}$ and each pivot $p \in \mathbb{P}$, we use them to compute the permutation for each u. The FHQTP stores all these permutations in the index, and after computing them, it discards the distances. Finally, we have a complete FHQT and each element has its permutation. The construction cost of FHQTP is the same as that of the classic FHQT.

If we have memory restrictions, we can save some space by not storing the central part of the permutations, because these permutants have a lesser contribution to the PBA prediction power. This was previously shown in [9].

4.2 Searching

To solve a range query (q, r), we divide the process in two steps. In the first one, we use the FHQTP as a standard FHQT in order to discard non-relevant objects. However, once a leaf is reached, instead of computing the distance d between all the objects stored in the leaf with the query, we put them in a candidate list (since they were not discarded by the FHQT). This process continues adding to the candidate list every non discarded object.

In the second step, we compute Π_q and we sort the candidate list in increasing order of permutation distance (e.g., Spearman Rho or Footrule), in a similar way as the standard PBA, but only considering the resulting list of candidates. Thanks to PBAs prediction property, it suffices to review just a small percentage of the top of the list.

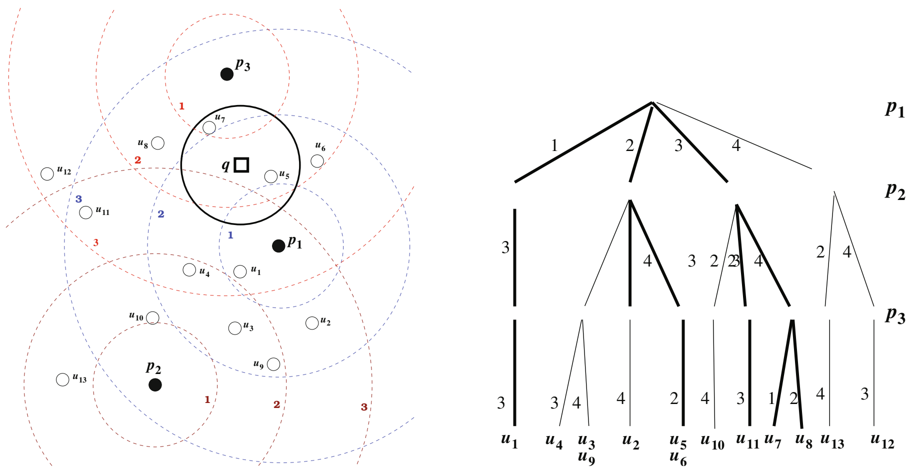
4.3 Example

Figure 1 depicts an example of our technique. This database consists of the small circles and we use the Euclidean distance. Pivots/permutants are the black filled circles. The circle in bold line centered at q represents a range query. Each circle in dash line, centered around pivots/permutants, represents a branch; and the elements inside it are part of the same branch. Note that for all pivots/permutants, we consider that every element beyond the third branch belongs to branch four. For example, object u_1 is placed in the first root branch as it is located at the first concentric circle from p_1 (notice that u_1 is the only one), and u_1 is in the third ring of both p_2 and p_3. The remaining elements are subject to the same process. Finally, the resulting FHQTP is composed by the FHQT shown in Fig. 1(b) and the corresponding permutations in Fig. 1(c).

In Fig. 1(a), the query q is located inside the rings 2, 4, and 2 of pivots/permutants p_1, p_2, and p_3, respectively; its permutation is $\Pi_q = 1\ 3\ 2$. Thin lines in Fig. 1(b) show the excluded branches. This means that elements u_1, u_5, u_6, u_7, u_8, and u_{11} are not discarded when solving the range query (q, r). However, according to their permutations (Fig. 1(c)), the permutations distances are $S_f(q, u_1) = 2, S_f(q, u_5) = 0, S_f(q, u_6) = 0, S_f(q, u_7) = 2, S_f(q, u_8) = 2$, and $S_f(q, u_{11}) = 4$. Therefore, the possible relevant objects can be reviewed in the following order: u_5, u_6, u_1, u_7, u_8, and u_{11}. Note that if we review a fraction (as authors describe in [6]) from the top of this list we have a good chance of retrieving the complete answer, and the chances increase as long as we review more non discarded elements.

4.4 Partial vs Full Permutation

Since we are processing the candidate list with PBA, we propose just to evaluate (and to keep) just a small part of permutations. According to [9], the most important part of permutations are the first and the last permutants. For example, if we divide each permutation in 3 parts and dismiss the central part, we use just 2/3 part of the permutations and we can get a good order to review



(a) Space. (b) FHQT.

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}
1 2 3	**1 2 3**	**1 2 3**	**1 2 3**	**1 3 2**	**1 3 2**	**3 1 2**	**3 1 2**	**1 2 3**	**2 1 3**	**2 1 3**	**3 1 2**	**2 1 3**

(c) Permutations.

Fig. 1. Sketch of our proposal.

the candidate list. In our example, in Fig. 1(c), we have $u_1 = 1\ 3, u_2 = 1\ 3, u_3 = 1\ 3, u_4 = 1\ 3, u_5 = 1\ 2$, and so on.

5 Experiment Results

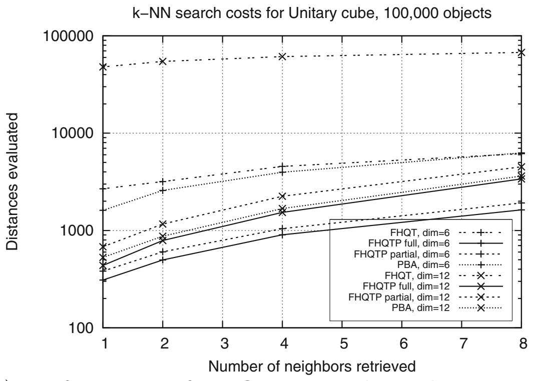
We tested our proposal using two kinds of metric databases. The first one is composed by several synthetic datasets that consists of uniformly distributed vectors in the unitary hypercube of dimensions 6 to 16 using the Euclidean distance. The second one consists of three real world datasets, namely, a set of English words using the Levenshtein's distance, a set of images from NASA, and a set of images from CoPhIR (Content-based Photo Image Retrieval). For the two image datasets we also use the Euclidean distance.

Since kNN queries are more appealing than range queries in practical applications, during the experimental evaluation, we simulate kNN queries with range queries using a radius that retrieves exactly the number of neighbors we need.

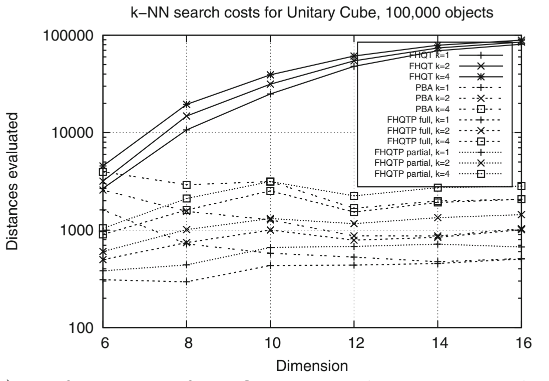
5.1 Synthetic Datasets

Each synthetic datasets is composed by 100,000 vectors uniformly distributed. These datasets allow us to control the intrinsic dimensionality of the space and analyze how the *curse of dimensionality* affects our proposal. We tested $kNN(q)$ queries with a query set of size 100 in dimensions that vary from 6 to 16.

The performance of our technique is shown in Fig. 2. Notice that our proposal computes less distances than the original FHQT technique. In high dimension, where FHQT is affected by the curse of dimensionality, using our proposal we can retrieve quickly the most similar objects. We repeated these experiments using different number of pivots, for $dim = 6$, we use $m = 6$, for $dim = 12$ we use $m = 12$, per each dimension we have 4 lines (FHQT, FHQTP full, FHQTP partial and original PBA). Our proposal made less distance computations than FHQT and PBA, and partial is better than FHQT but not than FHQT full.



(a) Performance of FHQTP over k in the unitary cube.



(b) Performance of FHQTP over dimension in the unitary cube.

Fig. 2. Comparison of search performance in the unitary cube synthetic spaces, 100,000 objects.

5.2 Real World Datasets

We use three real datasets. The first one is an English dictionary consisting of 69,069 words using the Levenshtein's distance, also known as edit distance. This

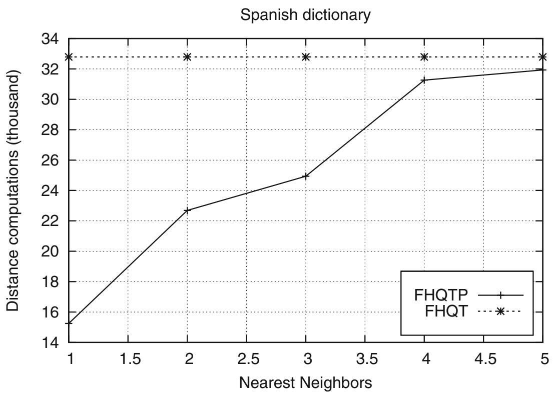
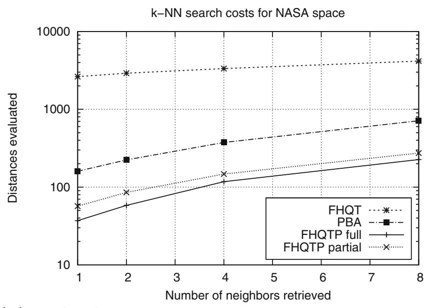
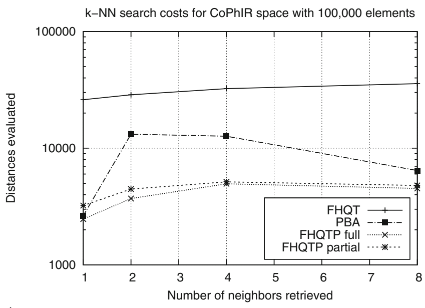


Fig. 3. Performance of FQTP using an English dictionary.



(a) NASA dataset average performance.



(b) CoPhIR dataset average performance.

Fig. 4. Performance of FHQTP on real database.

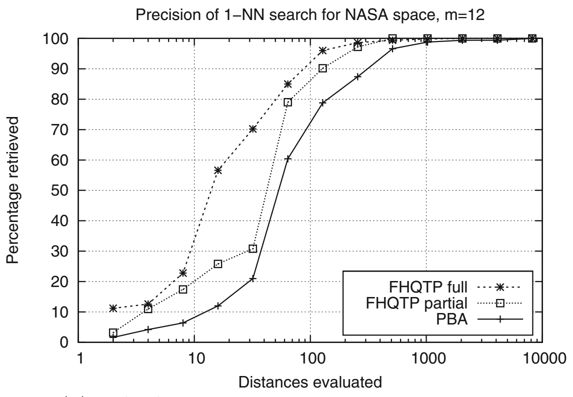
distance is equivalent to the minimum number of single character edit operations (character insertion, deletion or substitution) needed to convert one word into the other.

The second dataset contains 40,150 20-dimensional feature vectors, generated from images downloaded from NASA[1], where duplicated vectors were eliminated. Any quadratic form can be used as a distance, so we chose the Euclidean distance as the simplest, meaningful alternative.

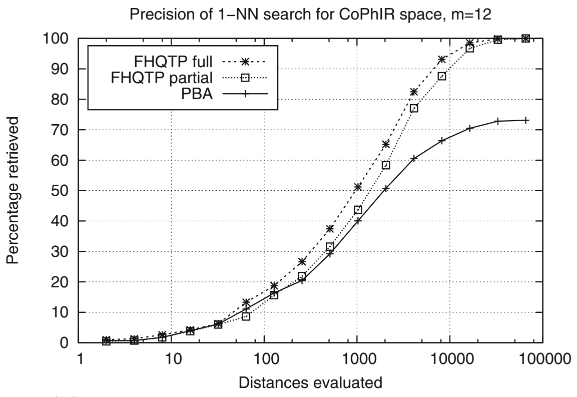
Finally, the third dataset is a subset of 100,000 images from CoPhIR [3]. For each image, the standard MPEG-7 image feature have been extracted. For the aforementioned reason, in this space we also use the Euclidean distance.

In Fig. 3, we show the performance of our technique for the English dictionary. Notice that our technique has an excellent performance for 1, 2, and 3 *NN*. Figure 4 shows how FHQT is not competitive in real databases. In the left side, Fig. 4(a) is for NASA database and, in the right side, Fig. 4(b) is for the CoPhIR

[1] At http://www.dimacs.rutgers.edu/Challenges/Sixth/software.html.



(a) NASA dataset average performance.



(b) CoPhIR dataset average performance.

Fig. 5. Performance of FHQTP on real databases, 1*NN* queries.

dataset. Both figures are showing the performance of our proposal for different values for k in kNN searches. We can get the answers up to 10x times faster. In this cases we use $m = 12$.

Finally, as shown in Fig. 5, we get the nearest neighbors faster than PBA when we review an incremental fraction, both for NASA database (Fig. 5(a)) and the CoPhIR dataset (Fig. 5(b)). Notice that we have a better performance than PBA in both databases.

6 Conclusions and Future Work

In this paper a new index is proposed. It consists in merging the best features of two known algorithms for metric spaces FHQT (Fixed Height Query Tree) and the Permutation based algorithm (PBA). Following the FHQT, when a query decides which branches are used and arrives at a leaf, all the objects in this leaf must be compared with the query. Instead, we can reuse the distance comparison performed, and to make the permutation per each object. At quering time, we use the FHQT to make a candidate list, then using the PBA technique, we review in a new order this list. This combination is a winner approach, it has an excellent performance because it works well in all dimensions.

For future work, we are interested in designing an algorithm for solving the k nearest neighbor query using our proposal, and extend this technique for other algorithms like FQA (Fixed Query Array).

References

1. Baeza-Yates, R.: Searching: an algorithmic tour. In: Kent, A., Williams, J. (eds.) Encyclopedia of Computer Science and Technology, vol. 37, pp. 331–359. Marcel Dekker Inc., New York (1997)
2. Baeza-Yates, R., Cunto, W., Manber, U., Wu, S.: Proximity matching using fixed-queries trees. In: Crochemore, M., Gusfield, D. (eds.) CPM 1994. LNCS, vol. 807, pp. 198–212. Springer, Heidelberg (1994). doi:10.1007/3-540-58094-8_18
3. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: a test collection for content-based image retrieval. CoRR abs/0905.4627v2 (2009). http://cophir.isti.cnr.it
4. Chávez, E., Figueroa, K.: Faster proximity searching in metric data. In: Monroy, R., Arroyo-Figueroa, G., Sucar, L.E., Sossa, H. (eds.) MICAI 2004. LNCS (LNAI), vol. 2972, pp. 222–231. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24694-7_23
5. Chávez, E., Figueroa, K., Navarro, G.: Proximity searching in high dimensional spaces with a proximity preserving order. In: Gelbukh, A., Albornoz, Á., Terashima-Marín, H. (eds.) MICAI 2005. LNCS (LNAI), vol. 3789, pp. 405–414. Springer, Heidelberg (2005). doi:10.1007/11579427_41
6. Chávez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) **30**(9), 1647–1658 (2009)
7. Chávez, E., Marroquín, J., Navarro, G.: Fixed queries array: a fast and economical data structure for proximity searching. Multimed. Tools Appl. (MTAP) **14**(2), 113–135 (2001)
8. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.: Proximity searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (2001)
9. Figueroa, K., Paredes, R.: An effective permutant selection heuristic for proximity searching in metric spaces. In: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Olvera-Lopez, J.A., Salas-Rodríguez, J., Suen, C.Y. (eds.) MCPR 2014. LNCS, vol. 8495, pp. 102–111. Springer, Cham (2014). doi:10.1007/978-3-319-07491-7_11
10. Samet, H.: Foundations of Multidimensional and Metric Data Structures. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco (2005)
11. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer, Heidelberg (2006)